# Dynamic Routing on OpenStreetMap Using Ant Colonies

Alexander Bertram[1,⋆] and Sebastian Iwanowski[2]

[1] TXS GmbH, Sonninstrasse 28, 20097 Hamburg, Germany
alexander.bertram@txs.de
[2] FH Wedel, University of Applied Sciences, Feldstr. 143, 22880 Wedel, Germany
iw@fh-wedel.de

**Abstract.** This paper presents the software AntScout for dynamic road navigation: The objective is to find the shortest path between arbitrarily chosen nodes in a road network considering the current traffic situation. AntScout reacts immediately to any change in the query or in the current traffic situation. This is achieved by the continuous application of ant colonies which are specially designed for problems with unexpected dynamic input change. The navigation tasks are performed on OpenStreetMap data which have been specially processed for that purpose.

AntScout provides an interface ready for use by third parties. It is published under an open source license. Modern implementation techniques such as the Scala actor concept are applied in order to make the software applicable for real world situations. Conceptually, some deficiencies of ant algorithms have been detected and repaired. The experiments with AntScout discovered the positive side effect that ant colony systems tend to toggle between several preferred routes nondeterministically when these routes are of similar quality. Thus, ant based navigation can also be used to distribute traffic uniformly among such routes without any further control actions.

**Keywords:** ant colonies, probabilistic approach, dynamic routing, OpenStreetMap, uniform traffic distribution.

## 1 Motivation

Nowadays, it is standard that cars have got a navigation device on board. For static conditions where the quality of roads is well-known and does not change the systems work fairly well.

Perhaps the greatest challenge is to keep the digital maps up-to-date with the real world. Another interesting topic of investigation is the question how to measure the quality of roads. This need not necessarily be the highest feasible maximum speed, but may also consider other criteria like road comfort, economic evaluations, interesting scenery, etc. In recent years, least effort of research has

⋆ This work was done while the author was working on his Master's degree at FH Wedel.

D. Pacino, S. Voß, and R.M. Jensen (Eds.): ICCL 2013, LNCS 8197, pp. 58–72, 2013.

been put into the algorithmic processing of the data available: The algorithms used work fairly well on present hardware even for big networks.

However, in the dynamic case, the situation is much different. By dynamic routing we understand the problem to find the best path between arbitrarily chosen points in the network considering the current quality of the road segments which is subject to dynamic and not predictable change. The major focus of application is to consider the current penetrability of the road as quality measure. The objective for this case is to find the shortest path considering the highest feasible speed for each road segment.

The question how to get the dynamic information for all road segments is not solved yet. Current technological approaches are Floating Car Data (FCD), Floating Phone Data (FPD) and Traffic Message Channel (TMC).

This work does not address the question how to get the dynamic data. The cited technologies make rapid progress. As an example, we refer to the Google FPD approach which shows a partially rather up-to-date traffic situation even for city streets. We assume that this problem will be solved with sufficient accuracy within a couple of years. Instead, we ask a different question: How do we process the huge amount of dynamic data such that we always have an up-to-date answer for the currently best route?

Note that the answer to this problem is not trivial, even if dynamic information is readily available for each road segment because the answer for the currently best route may be influenced by any road segment "between" start and destination where "between" may even consider deviations towards a different direction.

Ant algorithms have been specially designed to answer routing problems for such dynamic scenarios. They have already been successfully applied to computer networks and logistic problems. This work investigates how they behave in real road scenarios.

Also other work (e.g. [11]) has already investigated this question. However, the other authors did not work with real digital maps, but artificial maps. We decided to use OpenStreetMap (OSM) data, because the open standard of OSM makes it easy to extract the data we need and to insert any new dynamic information. Furthermore, the wiki approach of OSM gives hope that OSM will also solve the entry question addressed above with best quality, that is, how to keep the digital map up-to-date with reality.

## 2 State-of-the-Art and Its Limitations

### 2.1 Routing in State-of-the-Art Navigation Systems

Routing in static navigation systems is usually performed on a map which is available on-board. The algorithms used follow the basic idea of Dijkstra's algorithm which is easy to implement (the original is found in [6]). That algorithm is even theoretically optimal for networks with a constantly bounded number of neighbors for each junction (which is a realistic assumption).

Several improvements of the original algorithm such as the A* algorithm as well as special preprocessings of maps make it affordable to compute the optimal answer to each query within a very short response time. For a fairly recent analysis see [12].

Even small on-board-devices containing maps with more than 100000 junctions get the answer within fractions of a minute. This is why traditional navigation systems start computation only after the query is asked.

Nowadays, dynamic information referring to the current status of the road network is also provided. The most popular method applied in nearly all devices is a repeated update of information about traffic conditions on highways via Traffic Message Channel: The updated information is plugged into the local map of the on-board-device which recomputes the route for a target the vehicle is currently heading to. Computationally, this is feasible only, because the dynamic information provided is very scarce in time and does not apply to all road segments. Thus, recomputation does not occur very often. Since it is normally used for highways only, the next junction ahead where the old and the new answer may differ, is normally far away which leaves enough time for the complete new computation.

However, the situation would be different if we admit real-time information for all road segments of the network (not only highways) and if we also apply dynamic routing for city navigation. Then a recomputation would not be feasible, because the update would have been needed more frequently, and the driver would have to react within seconds to a new situation. Furthermore, and this is the most important argument, it is not feasible to transmit updates about each road segment of the total road network every minute to each mobile device in order to be integrated into on-board computation.

This suggests that future dynamic road navigation must be computed off-board on a supercomputer (which may be a virtual one using cloud computing) being provided with the most recent information about the status of all road segments. For this scenario, the on-board navigation devices would not do the actual computation anymore. Instead, it would rather transmit the queries of the user to the supercomputer and receive the computed route from the supercomputer and display it to the driver. In fact, in the front end the driver would not see any difference to present systems. In order to be fault-tolerant to mobile transmission failures, the traditional functionality of the navigation device may be applied as backup solution.

Smartphones using Google or other digital maps already comply to this off-board principle. Google maps can already display the current traffic situation which is retrieved from the mobile cell-phone users. This can be achieved not only for highways but also for city traffic. However, Google apparently does not integrate the dynamic information into the optimization algorithm: Google shows several (usually 3) routes which are reasonable for normal traffic conditions. For these suggestions, Google also shows the current situation. But Google would not include alternative routes when all routes suggested are disadvantageous under current conditions. And even if Google did, the open question would be how the

continuously changing current status of the road segments is integrated into the optimization process.

### 2.2 How Ant Systems Perform Routing Tasks

Ant algorithms are explicitly designed to perform dynamic routing tasks. They resemble the behaviour of ant colonies in nature. The difference to classical routing algorithms is the following:

**Eager computing:** Ant systems compute routing information continuously between any potential query points. This is called the eager computing strategy: The result is already computed for the current situation before the query is asked.

**Off-board middleware:** Ant systems are performed on an off-board middleware which is the necessary prerequisite for passing current information gathered by thousands of individuals to thousands of applying users.

**Statistical compression of individual information:** The dynamic information obtained by individuals is not stored individually but collected in **pheromones** which are information chunks containing quality information for links with respect to a certain target. For each target, each link has got its own pheromone value. This considerably condenses the information individually obtained. Since the pheromones are target-oriented, the pheromones do not just evaluate single road segments, but the total route from the place they are stored to the desired target.

**Local information first:** The local use of pheromones enables the system to give the answer in which direction to proceed within fractions of a second even if the total route has not been computed yet. This even applies to dynamic changes that just occurred.

First, we describe general properties all artificial ant systems share. Ant systems operate on an underlying graph and solve dedicated constraint problems. These problems may be scheduling or path finding tasks or combinations thereof. In principle, any constraint problem may be solved with ant systems as long as the underlying graph is properly defined.

An (artificial) ant is a software unit. Such units are continuously generated over time by the ant system. Each ant uses the current state of data of the underlying graph, considers the current constraints to be solved at the time of its generation, and tries to find a single solution for this problem. Each ant is influenced by the pheromones of the graph links lying directly ahead. It will not automatically use the link with the best pheromone values, but rather decide that probabilistically: The better a pheromone value, the more likely an ant will use it. After having found a solution, each ant gives feedback modifying the pheromones attached to the links this ant has used for its solution. The amount of modification depends on the quality of the result discovered by the ant. Thus, the pheromones represent the collected memory of previous ants having used the respective edge. Subsequently generated ants are biased by these pheromones for

their own construction of a solution. This is how they resemble the behaviour of real ants.

Ant systems found quite a few applications in logistics (for a real life application, cf. [2]). In these applications, the ant systems solve different variants of the vehicle routing problem.

In the following, we confine to the application of path finding in navigation systems: In this application, ant systems use the normal road network as graph. The nodes are the junctions and the edges are the individual road segments. For each node, there is a routing matrix evaluating the quality of using a certain adjacent edge in order to reach a certain selected target. This corresponds to the standard computer network protocols where ant systems have already been successfully applied (cf. [4]). The quality measures of this matrix are the pheromones. Note that the pheromones attached to an edge are not the same as the current cost function for this edge, because the pheromones do not only consider this edge but also all edges possibly behind on the way to the target.

For each (source, target) pair, ants are continuously produced at their source node. The task of each ant is to find a path to the specified target node.

For each path finding, the probability that an ant selects a certain edge depends not only on the quality of pheromones attached so far, but also on some heuristic value, which may directly involve the graph's cost function. The tradeoff may be tuned by several parameters. In general, the mutual consideration of task oriented pheromones and network oriented heuristic values ensures a balance between the emergence of stable paths and the adaptivity to dynamic changes.

In navigation, the path finding and updating functionalities of ants are temporally interweaved. This enables a highly parallel process, but this requires a suitable software environment to support that. In other applications of ant algorithms like scheduling, the solution finding and updating phases are more separated.

After an individual ant has found a path to the target, the update of the pheromones works as follows: The pheromone values of the edges used are increased, and the pheromones of the edges not used (but adjacent to the nodes used) are decreased. The latter process resembles evaporation which has been proven very practical in nature, because this gives more recent data a higher influence than older data. However, the difference gap by which the pheromones of the edges used are increased and the others are decreased depends on the quality of the overall route: The longer the travel time, the smaller the difference gap. This technique results in a rather fast emerging of good routes used by the majority of the ants.

For further information about ant systems and the details of the formulae used, we refer to the standard literature such as [9], [7], [8], [13]. Also in the master's thesis [1] which is the long version of this paper, the formulae are elaborated explicitly.

### 2.3 OpenStreetMap

OpenStreetMap[1] is the largest free geographic database of the world which is rapidly growing. OpenStreetMap gathers and updates its content from volunteer helpers throughout the world. Thus, it applies the same principle as wikipedia. The number of registered helpers is almost 1 million already. This makes OpenStreetMap the most accurate geographic database among all digital maps at least in countries with a lot of digitally oriented people, e.g., in most areas of Europe. Unlike other maps like GoogleMaps, OpenStreetMap has a specific strength also in foot paths and areas beyond roads.

Unlike other free maps like GoogleMaps, OpenStreetMap does not only provide the maps, but also the geodata themselves for free use. This makes it very prone for applying new technologies which is the main reason why we chose this map. Meanwhile, several routing providers are linked to at least the German version of OpenStreetMap[2]. However, they deal with static routing only.

The data of OpenStreetMap is provided in XML which makes it very convenient to be processed by any programming language with a web service interface.

Open StreetMap provides the concepts of `node` and `way`: A `node` is an arbitrary point defined by its geographic coordinates. A `way` is defined by a sequence of nodes. This is used to model roads, borders and rivers and any other object of 1 dimension.

`Road segments` are a special kind of `ways`. They are classified in several categories: The highest category is `motorway` and the lowest `residential`. In between there are the categories `primary`, `secondary`, `tertiary` and some more categories.

### 2.4 Conceptual Challenges

Ant systems are based on the eager computation principle: Each route query is computed in advance. Since the system must be prepared to answer any query, it must continuously update the best route between any two query points. This results in a number of queries that is quadratic in the number of nodes of the underlying graph. In order to ensure fast convergence of the pheromones to any dynamic situation, this quadratic number of queries must be repeated frequently within a minute.

In principle, this results in a run time that is a quadratic multiple of the run time of a single query. But there is hope to reduce this by parallelization and reduction of query nodes.

## 3 AntScout - Applying Ant Colonies on OpenStreetMap

AntScout is an open source software[3] published under the apache license version 2.0.

[1] `http://www.openstreetmap.org/`
[2] `http://www.openstreetmap.de/`
[3] `https://github.com/abertram/AntScout`

AntScout consists of two parts:

1. A preprocessing unit that extracts data from OpenStreetMap in order to enable the functionalities of the run time unit
2. A run time unit providing two functionalities:
   **user functionality:** performing navigation queries on the preprocessed map
   **operator functionality:** changing the feasible speed of selected road segments

The user functionality of the run time unit is the benchmark for the ant colony solution and the main part to be evaluated. The operator functionality corresponds to a traffic simulator and resembles dynamic behaviour, because this is the purpose for which we are using ant colonies.

### 3.1 Software Environment

As programming environment we used Scala which is a language integrating the principles of object-oriented and functional programming. The compiler produces Java byte code which may be executed on a Java Virtual Machine. Thus, Scala bears all advantages of Java as a programming language.

The reason for applying rather Scala than Java is its ability to provide parallelization: In contrast to Java's thread concept, Scala provides the actor concept of Erlang which is specially designed for parallelization. Originally, Scala had got an own actor library for that, but we rather used an open source framework called Akka[4]. In the meantime, Akka has been integrated in Scala's next version. Thus, our implementation is safe for future extensions. The strength of Akka is the smooth support of distributing parallel processes to several computers which enables cloud computing.

### 3.2 Preprocessing Functionality

The aim of the preprocessing functionality is to extract a network of nodes and links that is eligible for routing. In principle, each OpenStreetMap node that is part of a road segment may be a network node. But in order to reduce the complexity of the run time component, we reduce the number of nodes considerably by the following:

1. If an OpenStreetMap node is in between a road segment or connecting exactly two road segments, routing does not have a choice between alternatives. This is why we omit all such nodes and only consider nodes which are part of at least three road segments. Using graph terminology, we only consider nodes of degree at least 3.
2. We restrict to certain geographic areas using a rectangle placed upon OpenStreetMap data: Only nodes within this rectangle are considered, and only links between such nodes are considered, i.e., links stretching out of the rectangle are not considered.
3. We confine to predefined categories of road segments: Only road segments of this category or higher are considered.

[4] `http://akka.io/`

In order to efficiently reduce the number of nodes, the preprocessing unit combines these three principles simultaneously, i.e. it uses a predefined admissible rectangle and a predefined admissible lowest category and considers as admissible nodes only the nodes that are part of at least three road segments of that category or higher. A link in our graph is identified with a sequence of OSM road segments of the admissible categories. Note that the network obtained from this also contains nodes of degree 1 and 2, because some of the links incident to admissible nodes (which originally were all of degree at least 3) may stretch out of the admissible rectangle and, thus, have to be removed. For further reduction, we also merge two links meeting at a node of degree 2, thus avoiding nodes of degree 2. However, we leave nodes of degree 1 in the network which are dead ends.

The admissible rectangle may be arbitrarily chosen, and the admissible lowest categories are `motorway`, `motorway-link`, `trunk`, `trunk-link`, `primary`, `primary-link`, `secondary`, `secondary-link` and `tertiary`.

We applied the preprocessing to several areas of the town of Hamburg which is a densely populated city. Our reduction principles above applied to rectangles with areas up to 50 $km^2$. Using tertiary roads as lowest admissible categories provided a maximum of 150 nodes.

In off-city areas the area covered may be much higher, but we wanted to demonstrate that our dynamic routing software may even be applied in cities which we consider the hardest task due to the density of alternatives and the need for fast reaction to dynamic changes.

### 3.3 Run Time Functionality

Since our system should show how ant systems react to spontaneous traffic changes, we considered it crucial to integrate user and operator mode in the same interface, i.e., there should be no explicit switching between them: An operator action should be executed at any time, and also a user query should be specified at any time. Both types may be interweaved in an arbitrary order. The operator action referring to some road segment remains valid until a new action is executed for the same road segment. A user query is valid until a new query is asked.

AntScout always displays the route using the best pheromones between the currently valid source and target. This is considered the currently suggested route.

This gave us an easy method to measure the response time to any dynamic change represented by an operator action: We simply needed to measure the time between operator entry and visible route change.

Note that a car driver who is identified with the user of our setting has nothing to do with the individual ants of the colony system which are anonymous software units and not displayed at the interface at all: The car drivers always get the currently best answer, but the ants make a probabilistic decision for their route. The latter guarantees that there will always be ants using unfavourable routes which is important for the quick detection of dynamic changes.

Figure 1 is a screen shot of the user interface illustrating how the run time component of AntScout works:

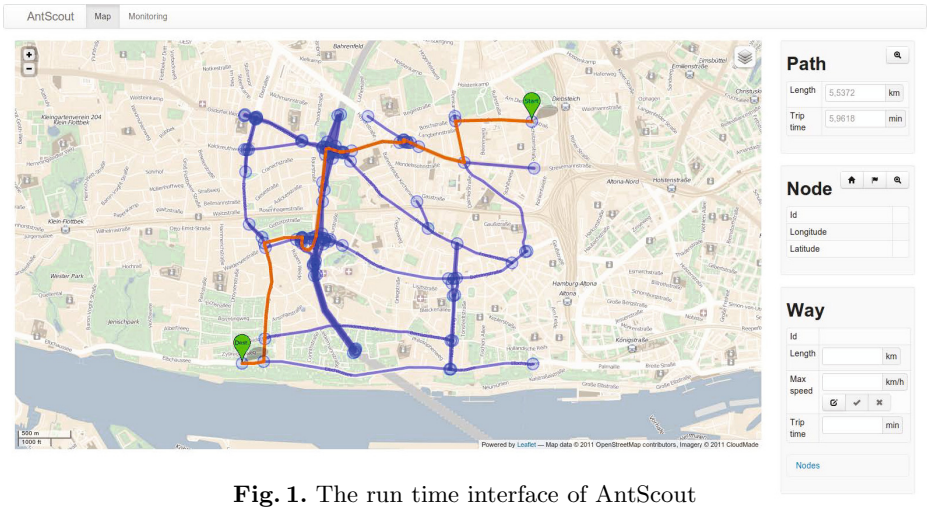


**Fig. 1.** The run time interface of AntScout

The user may select any node and make it to the source or target. AntScout will always show the currently best route in between. The interface entry **Path** shows the total time currently needed for the preferred route.

The operator may select any road segment: The interface shows the current maximum speed and the time for traversing the whole segment. The operator may change the maximum speed to any value simulating traffic jam or acceleration. The interface will respond with the new current traversing time in **Path** and possibly change the preferred route for the current user request.

### 3.4 Implementation Arrangements of AntScout

Due to the results of previous work at our department (cf. [14], English summary in [10]), we decided to implement the AntNet version which is the system described by Dorigo et al. [9]. For each node, AntNet works with a prepheromone matrix which is updated by the ants continuously and a pheromone matrix which is used by the ants for navigation. The prepheromone matrix is based on a statistical model using dynamic information only such as previous experiences of ants and flexible time windows. The pheromone matrix integrates static information into the values of the prepheromone matrix such as segment lengths. Both, prepheromone and pheromone matrices serve like a routing table for its node: There is a line for each possible target and a column for each adjacent node. A matrix entry (i,j) denotes the quality of using the link to adjacent node j if the target is node i.

The continuous generation of ants from each possible source node to each possible target node is the complexity bottleneck of the entire method. This is

why we did not model each ant as an independent actor for efficiency reasons. We rather modelled an ant to be a state of the nodes which are communicated and transformed among the neighbors. Thus, an ant colony system may be regarded a discrete state automaton. The handling of ants as passive states instead of active actors is a software acceleration which we see crucial for reducing the run time of a classical ant algorithm. Thus, an "ant" in our implementation is nothing else than a unit adding the current cost measures of the edges used until the target is reached.

### 3.5 Conceptual Improvements by AntScout

At initialization, the pheromone matrix has to start with some values. According to the theory, all prepheromones should start with the same values, because no ants have passed so far. If the segment lengths are added to that, this would result in a greedy strategy at the beginning.

We decided not to start with such a crude strategy and then let the ant colonies find suitable values statistically, but rather with initial values obtained from deterministic methods. Since we had to solve the all pairs shortest path problem, we did not apply Dijkstra's algorithm for each source, but rather applied the algorithm of Floyd-Warshall (for a modern description cf. [3], ch. 25.2) which is a little faster for that problem. This guarantees much faster convergence at the beginning.

Since the statistical path finding does not guarantee that each ant reaches its target at some time (e.g., the ant may get stuck in a loop), ants have to be removed after a while. [9] suggests to remove an ant, after the ant run into a loop. We first implemented this suggestion, but then found much better convergence when instead an ant is removed after a certain time has elapsed. For different networks, this time should depend on the size of the network.

Since the number of (source, target) pairs grows quadratic in the size of the network, it is important to control the number of ants generated for a given (source, target) pair. The trade-off is the following: The fewer ants are generated, the less computation load to the system (which is a trivial observation), but the more ants are generated in a certain time interval, the faster the expected convergence.

We applied the following approach to solve this dilemma: At each source $s$, ants are generated heading towards a predefined target $t$. Note that ants heading to a further distant target $t$, also pass other targets $t_i$ on their path from $s$ to $t$. The closer $t_i$ from $s$, the higher the probability that an ant originally heading to $t$ will also pass $t_i$. Catching up an original idea of Dorigo, Walther [10] already mentioned that the pheromones referring to targets $t_i$ should also be updated by ants heading to $t$ which would increase the accuracy of the pheromones referring to the trip from $s$ to $t_i$. Thus, if the generation frequency is the same for all (source, target) pairs, the accuracy for closer relations is much better than for further relations. This is exactly the opposite of what a user wants to see, because in closer relations the absolute quality difference between different routes is much less than for more distant relations. We compensate this by creating more ants

between more distant distance relations. For the implementation of this principle, we work with the following parameters:

**Distance groups:** The (source, target) pairs are collected into groups of "similar" distance. The distances are obtained using the static values computed at initialization.

**Launch interval:** This is the time interval between two single ants generated: The smaller the interval, the more ants are generated. In our implementation this interval is applied to the group of most distant pairs only.

**Launch delay:** This is the delay added to the launch interval for pairs of closer distance: The closer the distance, the greater the delay.

## 4 Tests and Results

Our test consisted of two parts: First, we did some experiments with parameters in order to adjust the scalability. Second, we evaluated the quality of AntScout from a user's point of view: How well and how fast does the ant colony react to a sudden change introduced by an operator.

The hardware on which we tested our software included an Intel Core i5-2520M processor with 4 x 2.5 GHz and a memory of 7.6 GB applied on a 64-bit system.

### 4.1 Adjusting the Scalability

In this investigation, we tested different values for launch interval and launch delay and tested their effect to quality which was measured by the number of ants who really reached their target and, thus, updated the pheromones. We compared this with the scalability of the system which was measured by the frequency in which a full garbage collection was started.

In a Java Virtual Machine, a full garbage collection is executed automatically when there are too many unreferenced objects. Such an event is dependent on the number of objects generated which in our case depends on the size of the network. The effect of a full garbage collection is an intermediate halt of the overall simulation. This has direct influence on the performance. This is why a full garbage collection is a good measure for scalability.

As an example, Fig. 2 shows the result for the map shown above consisting of 104 nodes.

Considering the dashed graph, it is a little surprising that the quality is not highest with the lowest launch interval. This effect is even more evident in larger maps. Our explanation is that a too frequent launch of ants is too fast for processing them, and this implies that quite a few of the generated ants will not reach their goal. On the other hand, the behaviour for larger launch intervals is expected because it is clear that fewer ants will reach their goal if fewer ants are generated.

Considering the solid graph, it is clear that a higher frequency of ant launch (corresponding to smaller launch intervals) will result in more garbage collection. This explains the monotonous decline of the corresponding graph. Since a

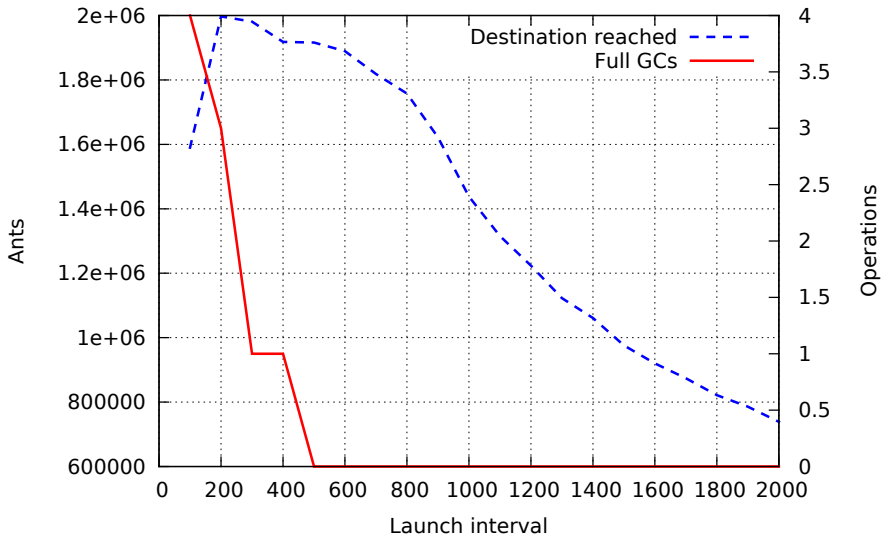


**Fig. 2.** Induced quality and scalability of ant generation

reasonable response time can only be achieved when there is no global garbage collection at all, we search for the minimum launch interval where global garbage collection does not occur. We denote this to be the *ideal interval*. In the respective diagram, this ideal interval length is 500 ms (where the solid graph intersects the abscissa).

We did similar investigations for other networks having the same number, less or more nodes. We achieved ideal launch intervals between 100 ms (for 61 nodes) and 1100 ms for 144 nodes. This meets our expectations that larger networks require longer launch intervals due to the quadratic increase of (source, target) pairs. For up to 150 nodes the quality of result (dashed graph) was still nearly optimum, but for larger networks it declined considerably. Thus, our approach successfully only applies to networks with up to 150 nodes, at least with the hardware we used.

For the launch delay, we showed in several tests for different launch intervals that it is best to keep the delay in a range of 20 % of the launch interval.

### 4.2 Evaluation of Results and Improving the Algorithm

The goal of our tests was to answer the following questions:

1. How does the system react when a preferred route deteriorates?
2. How fast does AntScout switch back into the initial state when a deteriorated route has improved again?
3. How does the system react when a so far nonpreferred route improves considerably such that now it would be a preferred candidate?

We tested this by a high variety of experiments to various (source, target) pairs in several maps. In many cases, the system behaved as wanted. The response time varied between instantly and less than 5 seconds. Only in some cases, we realized that the system got stuck in a nonoptimal route.

Careful analysis revealed the following: AntNet tends to prefer routes with fewer decision alternatives, i.e. paths with nodes of lower degree. This even holds when the selected route is considerably longer than the optimal one (more than 20 %).

By the following techniques, this behaviour can be avoided: Ants passing nodes with a higher degree get a higher relevance for updating the pheromones. An alternative or supplementing strategy is to make the number of generated ants dependent on the number of neighbors of the source: The more neighbors, the more ants are generated.

### 4.3 Oscillation

Due to the nondeterministic behaviour, the preferred routes oscillate between several candidates continuously.

We observed that such an oscillation only occurs when the routes do not differ more than 7 % from the average among all routes ever suggested. This implies that oscillating occurs only between routes that differ at most 14 %.

All experiments showed the following correlation: The more similar two routes were concerning their quality, the more oscillation occurred between them.

This is a reasonable behaviour which is easy to explain for a probabilistic algorithm. We suggest to utilize this for the following:

In real life, a driver who asked for a route should retain this route once he is suggested one. But the oscillating behaviour results in different routes for different drivers. Due to the observed correlation between frequency of oscillation and similarity of route qualities, this results in a perfect distribution of traffic among almost equal routes. Since this distribution is organized at random, we expect also a high social acceptance.

Thus, ant algorithms are very suitable for traffic management as a side effect.

## 5 Conclusion

This paper described the construction and evaluation of the software AntScout which applies ant colonies for routing in real road maps extracted from OpenStreetMap. AntScout is prototypic, but it offers an interface comfort that makes it testable by third parties.

AntScout proves that ant algorithms may be applied even without hierarchies in real maps with up to 150 nodes on a modern laptop. The response to sudden changes in the map (simulating traffic jam and its resolution) is rather fast (less than 5 seconds). The quality of the suggested routes is very reasonable: The simulation switches between the best route and alternatives that were within 14 per cent off the optimum.

Conceptually, the improvements to the state-of-the-art are the following:

1. We showed how to adapt OpenStreetMap maps such that they can be used for dynamic routing via ant colonies.
2. We showed that the published versions of ant algorithms tend to prefer routes with nodes of lower degrees and developed a concept how to avoid that.
3. We showed that the statistically fuzzy behaviour of ant algorithms can be used for traffic management with the goal of equally distributing the traffic. This makes congestions less likely to occur.

Next work to be done towards an integration into real navigation systems is the following:

1. The number of nodes extracted from OpenStreetMap can be further reduced: Quite a few OpenStreetMap nodes physically belong to the same junction. These nodes should be summarized to a supernode that serves as single source or target for the ants of the colony.
2. In order to enable a higher degree of parallelism, the ant simulations should be distributed to several computers. AntScout is perfectly prepared to this using the actor concept of the Akka library.
3. In order to improve the requirements of real navigation systems, AntScout should integrate other criteria than the mere passing time of road segments.
4. First attempts have already been published how to integrate hierarchies into ant algorithms conceptually (cf. [10], [5]). These suggestions should be elaborated and integrated into AntScout.

While the first three items are matters of implementation which do not require conceptual adjustments, the last item requires more conceptual research.

The physical visibility of the functionality of AntScout shows that a conceptually innovative and also very exotic paradigm such as ant colonies may be used for real world applications. We are convinced that navigation systems of the future will apply the following advantages of ant algorithms (elaborated in Section 2.2) regardless if they are called ant-based or not:

1. eager computing
2. off-board middleware
3. statistical compression of individual information
4. local information first

## References

1. Bertram, A.: Ant Scout - Dynamisches Routing auf OpenStreetMap. Master's thesis, FH Wedel (2012) (in German), `http://www.fh-wedel.de/fileadmin/mitarbeiter/iw/Abschlussarbeiten/MasterthesisBertram.pdf`
2. Blöcker, C., Iwanowski, S.: Utilizing an ant system for a competitive real-life planning scenario. In: COMPUTATION TOOLS 2012: Third International Conference on Computational Logics, Algebras, Programming, and Benchmarking, pp. 7–13 (2012)

3. Cormen, T., Stein, C., Leiserson, C., Rivest, R.: Introduction to Algorithms, 3rd edn. MIT Press (July 2009)
4. Di Caro, G., Dorigo, M.: Distributed stigmergetic control for communication networks. Journal of Artificial Intelligence Research 9, 317–365 (1998)
5. Dibowski, H.: Hierarchical routing system using ant based control. Master's thesis, TU Dresden (July 2003), http://www.kbs.twi.tudelft.nl/docs/MSc/2003/Dubowski_Henrik/thesis.pdf
6. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
7. Dorigo, M., Birattari, M., Stützle, T.: Ant colony optimization – artificial ants as a computational intelligence technique. IEEE Computational Intelligence Magazine 1, 28–39 (2006)
8. Dorigo, M., Gambardella, L.M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1, 53–66 (1997)
9. Dorigo, M., Stützle, T.: Ant colony optimization. The MIT Press (2004)
10. Iwanowski, S., Walther, T.: Dynamic road navigation with ant algorithms. Technical report, Fachhochschule Wedel (2008), http://www.fh-wedel.de/fileadmin/mitarbeiter/iw/Abschlussarbeiten/DynamicRoadNavigationWalther.pdf
11. Lerebourg, S., Dutot, A., Bertelle, C., Olivier, D.: Management of the road traffic by an ant algorithm. World Conference in Transport Research Society, vol. 9 (July 2004), http://scott.univ-lehavre.fr/~bertelle/publications/mas2003_7-road_traffic.pdf
12. Sanders, P., Schultes, D.: Engineering fast route planning algorithms. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 23–36. Springer, Heidelberg (2007), http://algo2.iti.kit.edu/documents/routeplanning/weaOverview.pdf
13. Taillard, É.D.: An introduction to ant systems. In: Laguna, M., González Velarde, J.L. (eds.) Computing Tools for Modeling, Optimization and Simulation, pp. 131–144. Kluwer, Boston (2000)
14. Walther, T.: Dynamische Fahrzeugnavigation auf Basis von Ameisenkolonien. Master's thesis, Fachhochschule Wedel (February 2006), http://www.fh-wedel.de/fileadmin/mitarbeiter/iw/Abschlussarbeiten/MasterarbeitWalther.pdf